PID in State Space Form

Garrett Wilson

May 13, 2016

Why?

You may have pondered the question of whether you would want to use PID or state space to control a system. If you want to compare how these two methods would perform on a certain system, you could rewrite your PID controller in state space form and easily run either controller on your system in a simulation in Matlab or Octave.

The only person I've found who has done this concluded the following after working out the math: "[T]here is no theoretical need to choose between PID or linear state-space control theory for purposes of system analysis. That choice might . . . need to be made because of other practical restrictions, such as the opportunity or difficulty for developing the necessary system model."

The Math

A PID controller normally looks like the following:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)d\tau + k_d \dot{e}(t)$$

We want to try rewriting this in state space form. Our state space form looks like:

$$\dot{x} = ax + bu$$
$$y = cx$$

Let $u = v + s$, the sum of our feedback component (v) and a set point or reference input (s). Then b_f is for the feedback and b_s for the setpoint:

$$\dot{x} = ax + b_f v + b_s s \tag{1}$$
$$y = cx \tag{2}$$

Let us define our error to be the difference between our output and our set point (the desired output):

$$e \equiv y - s = Cx - s$$

Now let's look at the three different components of the feedback term v that will in the end be summed together in our PID controller. For the proportional term:

$$v = -k_p e$$

For the integral term, let's define a new variable z whose derivative is the error e so that z is the integral of the error:

$$\dot{z} \equiv e$$
$$v = -k_i z$$

Finally, for the derivative,

$$v = -k_d \dot{y}$$

Note that we're using $\dot{y}$ rather than $\dot{e}$, but these will be equivalent except for when the set point s changes. It is preferable to use $\dot{y}$ since the derivative is defined even if s changes instantaneously. This is one of the approaches Wikipedia lists to get around step changes in s, another one being to never have a step change but gradually move between the old and new values.

We will use our model of the system to calculate $\dot{y} = c\dot{x}$:

$$\dot{y} = c\dot{x} = c(ax + b_f v + b_s s)$$
$$\implies v = -k_d c(ax + b_f v + b_s s)$$

Notice that we have a v on both the left and right sides of the equation. Solving for v:

$$\implies v + k_d c b_f v = -k_d c(ax + b_s s)$$
$$\implies (1 + k_d c b_f)v = -k_d c(ax + b_s s)$$
$$\implies v = -k_d (1 + k_d c b_f)^{-1} c(ax + b_s s)$$

And let's define K_g to make this easier to read:

$$K_g \equiv (1 + k_d c b_f)^{-1}$$
$$\implies v = -k_d K_g c(ax + b_s s)$$

Now let's sum all of these P, I, and D components of the feedback input v:

$$v = -k_p e - k_i z - k_d K_g c(ax + b_s s)$$

And define y_p, y_i, and y_d to be:

$$y_p \equiv e = \dot{z} = cx - s \tag{3}$$
$$y_i \equiv z \tag{4}$$
$$y_d \equiv K_g c(ax + b_s s) \tag{5}$$

Then, rewriting v in terms of these new variables:

$$v = -k_p y_p - k_i y_i - k_d y_d \tag{6}$$

We want to write all of this in state space. In addition to our original x state (or vector of states), let's add an additional state for z since the we need z in our PID controller for the integral term. For the state equation $\dot{X} = AX + BU$, using $\dot{z} = cx - s$ and equation 1:

$$\begin{bmatrix}\dot{x}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}a & 0\\ c & 0\end{bmatrix}\begin{bmatrix}x\\ z\end{bmatrix} + \begin{bmatrix}b_f & b_s\\ 0 & -1\end{bmatrix}\begin{bmatrix}v\\ s\end{bmatrix} \tag{7}$$

For the output $Y = CX + DU$, we want to not only get the system output y but also the three terms we'll be using in our PID control law. Using equations 2, 3, 4, 5:

$$\begin{bmatrix}y\\ y_p\\ y_i\\ y_d\end{bmatrix} = \begin{bmatrix}c & 0\\ c & 0\\ 0 & 1\\ K_g c a & 0\end{bmatrix}\begin{bmatrix}x\\ z\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & -1\\ 0 & 0\\ 0 & K_g c b_s\end{bmatrix}\begin{bmatrix}v\\ s\end{bmatrix} \tag{8}$$

Finally, we can write the control law:

$$v = -KY$$
$$K \equiv \begin{bmatrix}0 & k_p & k_i & k_d\end{bmatrix}$$

It would be nice if we could get something we could run through $lsim$ nicely. If we plug the control law back into our state space equations, rewriting it without the v input, using equations 6 and 7:

$$\begin{bmatrix}\dot{x}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}a & 0\\ c & 0\end{bmatrix}\begin{bmatrix}x\\ z\end{bmatrix} + \begin{bmatrix}b_f & b_s\\ 0 & -1\end{bmatrix}\begin{bmatrix}-k_p y_p - k_i y_i - k_d y_d\\ s\end{bmatrix}$$

Then, plugging in 3, 4, and 5:

$$\begin{bmatrix}\dot{x}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}a & 0\\ c & 0\end{bmatrix}\begin{bmatrix}x\\ z\end{bmatrix} + \begin{bmatrix}b_f & b_s\\ 0 & -1\end{bmatrix}\begin{bmatrix}-k_p(cx - s) - k_i z - k_d K_g c(ax + b_s s)\\ s\end{bmatrix}$$

$$\implies \begin{bmatrix}\dot{x}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}a & 0\\ c & 0\end{bmatrix}\begin{bmatrix}x\\ z\end{bmatrix} + \begin{bmatrix}b_f & b_s\\ 0 & -1\end{bmatrix}\begin{bmatrix}-k_p(cx - s) - k_i z - k_d K_g c(ax + b_s s)\\ s\end{bmatrix}$$

With a little bit more algebra, we get:

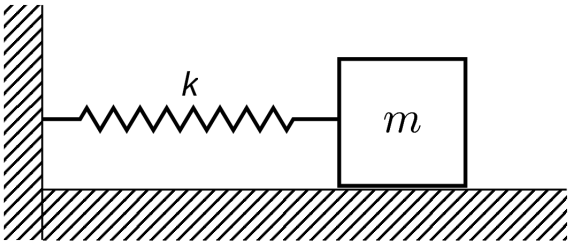
$$\begin{bmatrix}\dot{x}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}a - b_f(k_p c + k_d K_g c a) & -b_f k_i\\ c & 0\end{bmatrix}\begin{bmatrix}x\\ z\end{bmatrix} + \begin{bmatrix}b_f(k_p - k_d K_g c b_s) + b_s\\ -1\end{bmatrix} s \tag{9}$$

For the output, let's just look at the original y as in equation 2:

$$y = \begin{bmatrix}c & 0\end{bmatrix}\begin{bmatrix}x\\ z\end{bmatrix} \tag{10}$$

Example

Let's try out a PID controller on a second-order system of a mass, spring, and a force applied to the right on the mass.



Writing the state space equations $\dot{w} = Aw + Bu$ and $y = Cw = Du$:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} F$$

$$y = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$$

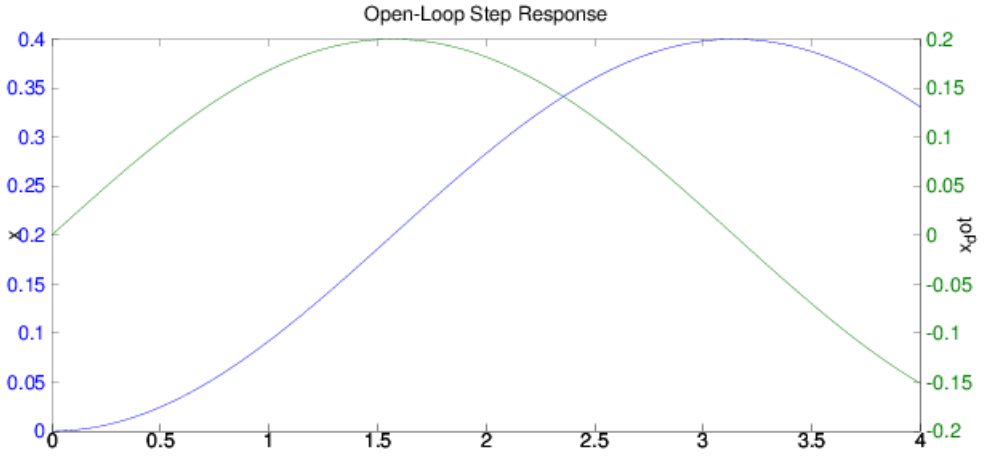
So,

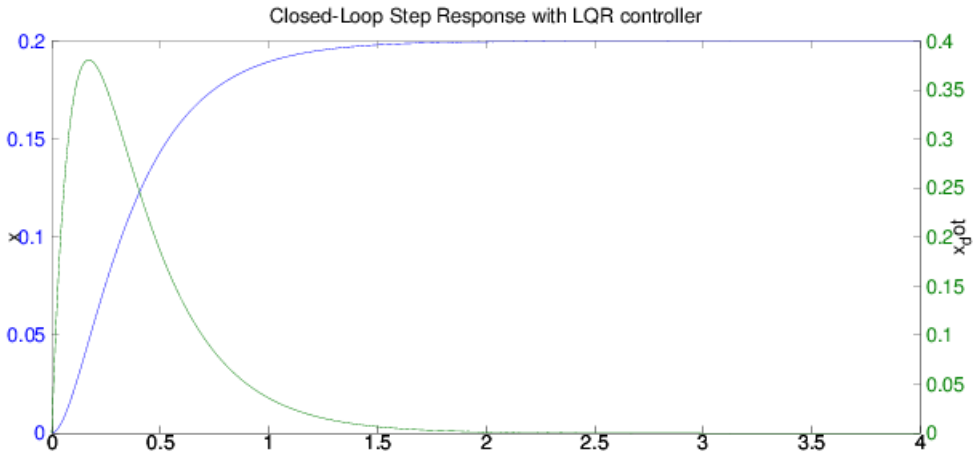
$$A = \begin{bmatrix} 0 & 1 \\ -k/m & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 1/m \end{bmatrix}, \qquad C = I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad D = 0$$

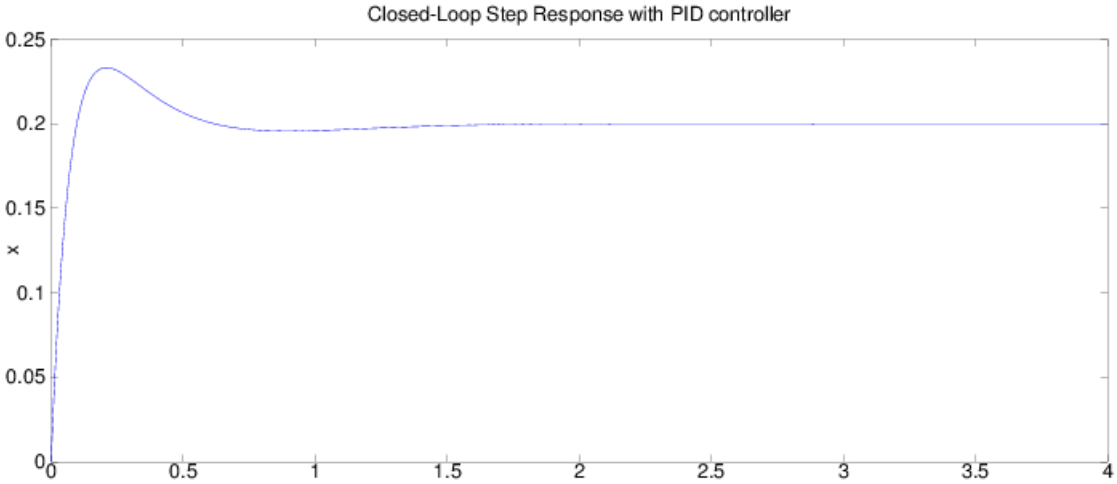
When converting our PID controller to state space, we'll have to make the C matrix only have a single output (it may be possible to do more, but so far I have only made it work with a single output):

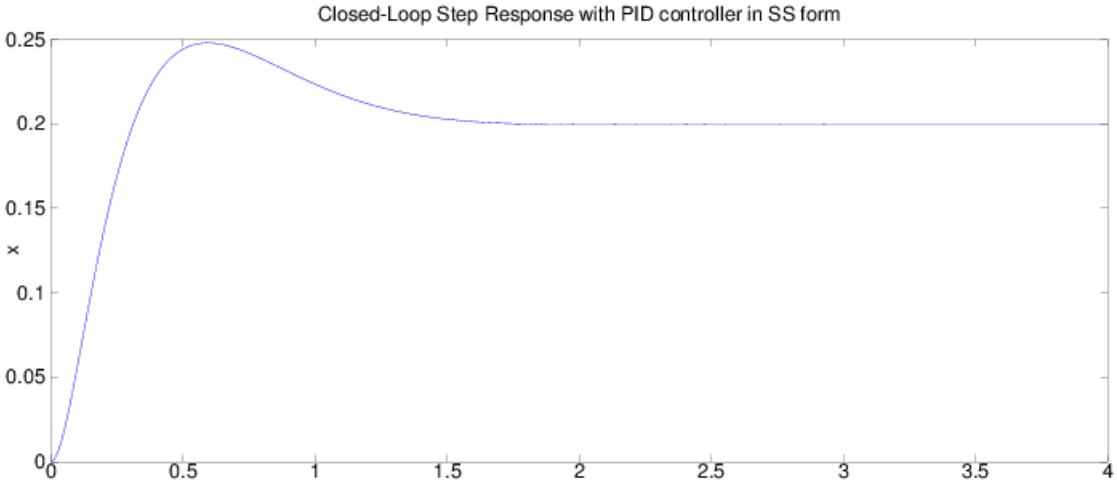
$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Below is the open-loop response and the response with an LQR, PID, and the PID in state space form controllers.









You'll notice the PID in state space form response is different from the PID response. We expect it to be somewhat different because in the straight PID controller, the derivative is not calculated based on the system model.

Note that in this case I set:

$$b_f = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

If b_f is just zeros, then the output is far from what the straight PID controller response is. I have yet to fully understand b_f.

Octave Code for Example

```
pkg load control;

% Plot both x and x_dot
function plotResponse(sys, plotTitle)
figure;
t = 0:0.01:4;
r = 0.2*ones(size(t));
[y,t,x]=lsim(sys,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','x');
set(get(AX(2),'Ylabel'),'String','x_dot');
title(plotTitle)
end

% Plot only x
function plotResponseSingle(sys, plotTitle)
figure;
t = 0:0.01:4;
r = 0.2*ones(size(t));
[y,t,x]=lsim(sys,r,t);
plot(t,y,'-b');
ylabel('x');
title(plotTitle)
end

% State space for simple second-order spring-mass equation
k = 1;
m = 1;

A = [0 1; -k/m 0];
B = [0; 1/m];
C = eye(2);
D = 0;

states = {'x' 'x_dot'};
inputs = {'F'};
outputs = {'x'; 'x_dot'};
sys_ss = ss(A,B,C,D,
    'statename',states,
    'inputname',inputs,
    'outputname',outputs);
plotResponse(sys_ss, 'Open-Loop Step Response');
print -dpng -S"700,300" -F"Helvetia:6" image-ol.png
```

```
44
45 % Controller using LQR
46 Q = C'*C;
47 Q(1,1) = 10;
48 R = 0.01;
49 K = lqr(A,B,Q,R);
50
51 % Correct position error
52 Cn = [1 0];
53 sys_nbar = ss(A,B,Cn,0);
54 % rscale from: http://ctms.engin.umich.edu/CTMS/index.php?aux=Extras_rscale
55 Nbar = rscale(sys_nbar,K);
56
57 Ac = [(A-B*K)];
58 Bc = [B*Nbar];
59 Cc = [C];
60 Dc = [D];
61
62 sys_cl = ss(Ac,Bc,Cc,Dc,
63     'statename',states,
64     'inputname',inputs,
65     'outputname',outputs);
66 plotResponse(sys_cl,
67     'Closed-Loop Step Response with LQR controller');
68 print -dpng -S"700,300" -F"Helvetia:6" image-lqr.png
69
70 % Use a PID controller
71 Kp = 100;
72 Ki = 200;
73 Kd = 20;
74
75 % We need to have SISO, so redefine C to only give us x out
76 C_siso = [1 0];
77 outputs_siso = {'x'};
78 sys_ss_siso = ss(A,B,C_siso,D,
79     'statename',states,
80     'inputname',inputs,
81     'outputname',outputs_siso);
82 sys_tf = tf(sys_ss_siso);
83
84 pid_controller = pid(Kp,Ki,Kd);
85 sys_cl_pid = feedback(pid_controller*sys_tf,1);
86 plotResponseSingle(sys_cl_pid,
87     'Closed-Loop Step Response with PID controller');
88 print -dpng -S"700,300" -F"Helvetia:6" image-pid.png
89
90 % Now let's use our new PID in SS form controller
91 %
92 % Note 1: We're using C_siso since with a PID controller you only have one
93 % output.
94 %
95 % Note 2: bf = [0;0] (and thus Kg = 1) since normally feedback does not couple
96 % directly and instantaneously into the output, but then it doesn't work
97 bf = B;
```

```
bs = B;
Kg = inv(1 + Kd*C_siso*bf);

% u = [v;s]
Apid = [A zeros(size(A,1),1); C_siso zeros(1,1)];
Bpid = [bf bs; 0 -1];
Cpid = [C_siso 0; C_siso 0; zeros(1,size(C,2)) 1; Kg*C_siso*A 0];
Dpid = [0 0; 0 -1; 0 0; 0 Kg*C_siso*bs];

% u = s, allowing us to run this nicely in lsim
Apid_cl = [A-bf*(Kp*C_siso+Kd*Kg*C_siso*A) -bf*Ki; C_siso 0];
Bpid_cl = [bf*(Kp-Kd*Kg*C_siso*bs)+bs; -1];
Cpid_cl = [C_siso 0];
Dpid_cl = 0;

states_pid = {'x' 'x_dot' 'z'};
inputs_pid = {'s'};
sys_ss_pid = ss(Apid_cl,Bpid_cl,Cpid_cl,Dpid_cl,
    'statename',states_pid,
    'inputname',inputs_pid,
    'outputname',outputs_siso);
plotResponseSingle(sys_ss_pid,
    'Closed-Loop Step Response with PID controller in SS form');
print -dpng -S"700,300" -F"Helvetia:6" image-pid-ss.png
```

Sources

A significant portion of the math comes from here, but I used $\dot{z} = Cx - s$ rather than $\dot{z} = b_e y_p$ (at Frohne's suggestion) since it made the end result look cleaner, and I continued on to find the A, B, C, and D matrices if the only input we have is the set point s rather than both v and s so we can simulate this with $lsim$:

http://home.earthlink.net/~ltrammell/tech/pidvslin.htm

Describes three approaches to deal with instantaneous step changes in the set point, one of which is by using $\dot{y}$ rather than $\dot{e}$:

https://en.wikipedia.org/wiki/PID_controller#Setpoint_step_change

Image of the spring-mass system:

https://minireference.com/_media/physics/mass_spring-highres.png